Optimistic Replication and Resolution

Marc Shapiro

▶ **To cite this version:**

Marc Shapiro. Optimistic Replication and Resolution. Özsu, M. Tamer; Liu, Ling. Encyclopedia of Database Systems, Springer-Verlag, pp.1995–1995, 2009, 10.1007/978-0-387-39940-9_258 . hal-01248202

HAL Id: hal-01248202
https://inria.hal.science/hal-01248202

Submitted on 24 Dec 2015

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Optimistic Replication and Resolution

Marc Shapiro
INRIA Paris-Rocquencourt & LIP6, http://www-sor.inria.fr/~shapiro/

January 29, 2009

1 Synonyms

Optimistic Replication, OR, Reconciliation-based Data Replication, Lazy Replication, Multi-Master System.

The term "Optimistic Replication" is prevalent in the distributed systems and distributed algorithms literature. The database literature prefers "Lazy Replication."

2 Definition

Data replication places physical copies of a shared logical item onto different sites. *Optimistic replication* (OR) allows a program at some site to read or update the local replica at any time. An update is *tentative* because it may conflict with a remote update. Such conflicts are resolved after the fact, in the background. Replicas may *diverge* occasionally but are expected to converge eventually (see entry on EVENTUAL CONSISTENCY).

OR avoids the need for distributed concurrency control prior to using an item. It allows a site to execute even when remote sites have crashed, when network connectivity is poor or expensive, or while disconnected from the network. *Disconnected operation,* the capability to compute while disconnected from a data source, e.g., in mobile computing, requires OR. In *computer-supported co-operative work*, OR enables a user to temporarily insulate himself from other users.

3 Historical background

The first historical instance of OR is Johnson's and Thomas's replicated database (1976).[1]

Usenet News (1979) was an important and inspirational development. News supports a large-scale ever-growing database of (read-only) items, posted by users all over the world. A Usenet site connects infrequently (e.g., daily) with

[1] The vocabulary used in this history is defined in Section 4.

its peers. New items are flooded to other sites, and are received in arbitrary order. Users occasionally observe ordering anomalies, but this is not considered a problem. However, system administrators must deal manually with conflicts over administrative operations.

In 1984, Wuu and Bernstein's replicated mutable key-value-pair database uses an operation log, transmitted by an *anti-entropy* protocol: site A sends to site B only the tail of A's log that B has not yet seen [12]. Concurrent operations either commute or have a natural semantic order; non-concurrent operations execute in happens-before order.

The Lotus Notes system (1988) supports co-operative work between mobile enterprise users. It replicates a database of discrete items in a peer-to-peer manner. Notes is state-based, and uses a Last-Writer Wins policy. A deleted item is replaced by a *tombstone*.

Several file systems, designed in the early 1990s to support disconnected work, e.g., Coda [3], are state based and use version vectors for conflict detection. Conflicts over some specific object types (e.g., directories or mailboxes) cause automatic resolver programs to run. The others must be resolved manually.

Golding (1992) [1] studies a replicated database of mutable key-value pairs. This system purges an operation from the log when it can prove that it was delivered to all sites. Consistency is ensured by defining a total order of operations.

Bayou (1994–1997) is an innovative general-purpose database for mobile users [6]. Bayou is operation-based and uses an anti-entropy protocol. Each site executes transactions in arbitrary order; transactions remain tentative. The eventual serialisation order is the order of execution at a designated *primary* site. Other sites roll back their tentative state, and re-execute committed transactions in commit order.

In 1996, Gray *et al.* argued that OR databases for disconnected work cannot scale [2], because conflict reconciliation is expensive, conflict probability rises as the third power of the number of nodes, and the wait probability further increases quadratically with disconnection time.

Breitbart *et al.* [**?**] describe a partially-replicated database that uses a form of OR. Each item has a designated primary site and may be replicated at any number of secondary sites. A read may occur at a secondary site but a write must occur on the primary. It follows that a write transaction updates a single site. If transactions are serialisable at each site, and update propagation is restricted to avoid ordering anomalies, then transactions are serialisable despite lazy propagation.

The Computer-Supported Cooperative Work (CSCW) community invented (1989) a form of OR called Operational Transformation (OT). Conflicting operations are *transformed*, by modifying their arguments, in order to execute in arbitrary order [9].

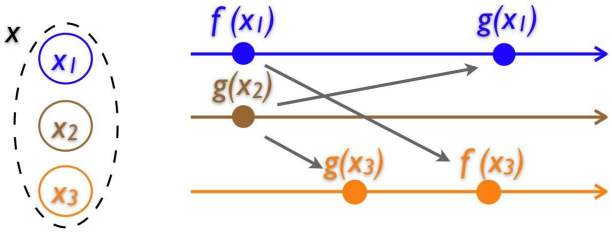

Figure 1: Three sites with replicas of logical item x. Site 1 initiates transaction f, Site 2 initiates g. The system propagates and replays on remote sites. Site 3 executes in the order $g; f$, whereas Site 1 replays f before g. Eventually, Site 2 will also execute f.

4 Scientific fundamentals

Figure 1 depicts a logical item x, concretey replicated at three different sites. In OR, any site may *submit* or *initiate* a transaction reading or writing the local replica. If the transaction succeeds locally, the system *propagates* it to other sites, and *replays* the transaction on the remote sites, in a *lazy* manner, in the background. Local execution is *tentative* and may be rolled back later, because of a *conflict* with a concurrent remote transaction.[2]

OR is opposed to pessimistic (or eager) replication, where a local transaction terminates only when it commits globally. Pessimistic replication establishes a total order for committed transactions, at the latest when each transaction terminates. In contrast, OR generally relaxes the ordering requirements and/or converges to a common order *a posteriori*. The effects of a tentative transaction can be observed, thus OR protocols may violate the *isolation* property and allow cascading aborts and retries to occur.

4.1 Transmitting and replaying updates

In OR, updates are propagated lazily, in the background, after the transaction has terminated locally. Transmission usually uses peer-to-peer epidemic or anti-entropy techniques (see entry on Peer-to-Peer Content Distribution).

A site that receives a remote update *replays* it, i.e., incorporates it into the local replica. There are two main approaches. In the *state-based* approach, the initiator site transmits the after-values of the transaction, and other sites assign the after-value to their local replica. In the *operation-based* approach, the initiator sends the program of the transaction itself, and other sites re-execute the transaction.

State-based replay is guaranteed to be deterministic. State-based replay can be more efficient, since the replay code is just a write. On the downside, if the granularity is large, then state-based transmission is expensive and replay

[2]The happens-before and concurrency relations are defined formally by Lamport [4]. Transaction A happens-before B, if B was initiated on some site after A executed at that site. Two transactions are concurrent if neither happens-before the other.

is subject to false conflicts. Furthermore, logical operations are more likely to commute than writes, thus operation-based replay typically causes fewer aborts.

The defining characteristic of OR is that any synchronisation between sites occurs in the background, after local termination, i.e., off the critical path of the application.

4.2 Conflicts

Each transaction taken individually is assumed correct (the C of the ACID properties), i.e., it maintains semantic invariants. For example, ensuring that a bank account remains positive, or that a person is not scheduled in two different meetings at the same time.

As is clear from Figure 1, concurrent transactions may be delivered to different sites in different orders.[3] However, consistency requires that local schedules be equivalent. In this respect, one may classify pairs of concurrent transactions as commuting, non-commuting, and antagonistic. Transactions *conflict* if they are mutually non-commuting or mutually antagonistic.

The relative execution order of *commuting* transactions is immaterial; they require no remote synchronisation. Formally, two transactions T_1 and T_2 commute if execution order $T_1; T_2$ returns the same results to the user and leaves the database in the same state as the order $T_2; T_1$. For instance, depositing €10 in a bank account commutes with a depositing €20 into the same account, and also commutes with withdrawing €100 from an independent account.

If running concurrent transactions together would violate an invariant, they are said *antagonistic*. Safety requires aborting one or the other (or both). For instance, if T_1 schedules me in a meeting from 10:00 to 12:00, and T_2 schedules a meeting from 11:00 to 13:00, they are antagonistic since no combination of both T_1 and T_2 can be correct.

If two transactions are *non-commuting* and neither is aborted, then their relative execution order must be the same at all sites. Consider for instance T_1 = "transfer balance to savings" and T_2 = "deposit €100". Both orders $T_1; T_2$ and $T_2; T_1$ make sense, but the result is clearly different. There must be a system-wide consensus on the order chosen.

4.3 Conflict resolution and reconciliation

Conflict resolution rewrites or aborts transactions to remove conflicts. Conflict resolution can be either manual or automatic. Manual conflict resolution simply allows conflicting transactions to proceed, thereby creating conflicting versions; it is up to the user to create a new, merged version.

Reconciliation detects and repairs conflicts, and combines non-conflicting updates. Thus transactions are *tentative*, i.e., a tentatively-successful transaction may have to roll back for reconciliation purposes. OR resolves conflicts *a posteriori* (whereas pessimistic approaches avoid them *a priori*).

[3] Dependent transactions are assumed to execute in dependency order; see Section 4.7.

In many systems, data invariants are either unknown or not communicated to the system. In this case, the system designer conservatively assumes that concurrent transactions that access the same item, and one writes the item (or both), then they are are antagonistic. Then, one of them must abort, or both.

A few systems, such as Bayou [11] or IceCube [7] support an application-specific check of invariants.

4.4 Last Writer Wins

When transactions consist only of writes, a common approach is to ensure a global precedence order.

For instance, many replicated file systems follow the "Last Writer Wins" (LWW) approach. Files have timestamps that increase with successive versions. When the file system encounters two concurrent versions of the same file, it overwrites the one with the smallest timestamp with the "younger" one (highest timestamp). The write with the smallest timestamp is lost; this approach violates the Durability property of ACID.

4.5 Semantic resolvers

A resolver is an application-specific conflict resolution program that automatically merges two conflicting versions of an item into a new one. For exampe, the Amazon online book store resolves problems with a user's "shopping cart" by taking the union of any concurrent instances. This maximizes availability despite network outages, crashes, and the user opening multiple sessions.

A resolver should ensure that the conflicting transactions are made to commute. In a state-based approach, a resolver generally parses the item's state into small, independent sub-items. Then it applies a LWW policy to updated and tombstoned sub-items, and a union policy to newly-created sub-items.

The most elaborate example exists in Bayou. A Bayou transaction has three components: the dependency check, the write, and the merge procedure. The former is a database query that checks for conflicts when replaying. The write (a SQL update) executes only if the consistency check succeeds. If it fails, the merge procedure (an arbitrary but deterministic program) provides a chance to fix the conflict. However, it is very difficult to write merge procedures in the general case.

4.6 Operational Transformation

In Operational Transformation (OT), conflicting operations are *transformed* [9]. Consider two users editing the shared text `"abc"`. User 1 initiates `insert("x", 2)` resulting in `"aXbc"` and User 2 initiates `delete(3)`, resulting in `"ab"`. When User 2 replays the insert, the result is `"aXb"` as expected. However for User 1 to observe the same result, the delete must be re-written to `delete(2)`.

In essence, the operations were specified in a non-commuting way, but transformation makes them commute. OT assumes that transformation is always

possible. The OT literature focuses on a simple, linear, shared edit buffer data type, for which numerous transformation algorithms have been proposed.

OT requires two correctness conditions, often called TP1 and TP2. TP1 requires that, for any two concurrent operations A and B, running "A followed by {B transformed in the context of A}" yield the same result as "B followed by {A transformed in the context of B}". TP1 is relatively easy to satisfy, and is sufficient if replay is somehow serialised.

TP2 requires that transformation functions themselves commute. TP2 is necessary if replay is in arbitrary order, e.g., in a peer-to-peer system. The vast majority of published non-serialised OT algorithms have been shown to violate TP2 [5].

4.7 Scheduling transactions content and ordering

In order to capture any causal dependencies, transactions execute in happens-before order. As explained in Section 4.2, antagonistic transactions cause aborts, and non-commuting transactions must be mutually ordered. This so-called *serialisation* requires a consensus.

Whereas pessimistic appoaches serialise *a priori*, most OR systems execute execute transactions tentatively in arbitrary order and serialise *a posteriori*. Some executions are rolled back; cascading aborts may occur.

A prime example is the Bayou system [11]. Each site executes transactions in the order received. Eventually, the transactions reach a distinguished *primary* site. If a transaction fails its dependency check at the primary, then it aborts everywhere. Transactions that succeed commit, and are serialised in the execution order of the primary.

The IceCube system showed that it is possible to improve the user experience by scheduling operations intelligently [7]. IceCube is a middleware that relieves the application programmer from many of the complexities of reconciliation. Multiple applications may co-exist on top of IceCube. Applications expose semantic annotations, indicating which operation pairs commute or not, are antagonistic, dependent, or have an inherent semantic order. The user may create atomic groups of operations from different applications. The IceCube scheduler performs an optimisation procedure over a batch of operations, minimising the number of aborted operations. The user commits any of the alternative schedules proposed by the system.

4.8 Freshness of replicas

Applications may benefit from *freshness* or quality-of-service guarantees, e.g., that no replica diverges by more than a known amount from the ideal, strongly-consistent state. Such guarantees come at the expense of decreased availability.

The Bayou system proposes qualitative "session guarantees" on the relative ordering of operations [10]. For instance, Read-Your-Writes (RYW) guarantees that a read observes the effect of a write by the same user, even if initiated at a different site. RYW ensures that immediately after changing his password, a

user can log in with the new password. Other similar guarantees are Monotonic-Reads, Writes-Follow-Reads, and Monotonic-Writes.

Systems such as TACT control replica divergence quantatively [13]. TACT provides a time-based guarantee, allowing an item to remain stale for only a bounded amount of time. TACT implements this by pushing an update operation to remote replicas before the time limit elapses. TACT also provides "order bounding," i.e., limiting the number of uncommitted operations: when a site reaches a user-defined bound on the number of uncommitted operations, it stops accepting new ones. Finally, TACT can bound the difference between numeric values. For this, each replica is allocated a quota. Each site estimates the progress of other sites, using vector clock techniques. The site stops initiating operations once its cumulative modifications, or the estimated remote updates to the item, reach the quota. At that point the site pushes its updates and pulls remote operations. For example a bank account might be replicated at ten sites. To guarantee that the balance observed is within €50 of the truth, each site's quota is €50/10 = €5. Whenever the difference estimated by a site reaches €5, it synchronises with the others.

4.9 Optimistic replication vs. optimistic concurrency control

The word "optimistic" has different, but related, meanings when used in the context of replication and of concurrency control.

Optimistic replication (OR) means that updates propagate lazily. There is no *a priori* total order of transactions. There is no point in time where different sites are guaranteed to have the same (or equivalent) state. Cascading aborts are possible.

Optimistic concurrency control (OCC) means that conflicting transactions are allowed to proceed concurrently. However, in most OCC implementations, a transaction validates before terminating. A transaction is serialised with respect to concurrent transactions, at the latest when it terminates, and cascading aborts do not occur.

5 Key applications

Usenet News pioneered the OR concept, allowing to share write-only information over a slow, but cheap network using dial-up modems over telephone lines.

Mobile users want to be able to work as usual, even when disconnected from the network. Thus, mobile computing is a key driver for OR applications. Systems designed for disconnected work that use OR include the Coda file system [3], the Bayou shared database [11], or the Lotus Notes collaborative suite.

Another important application area is Computer-Supported Collaborative Work. In this domain, users must be able to update shared artefacts in complex ways without interfering with one another. OR allows a user to insulate himself temporarily from other users. A key example is the Concurrent Versioning

System (CVS), which enables collaborative authoring of computer programs [**?**]. Bayou and Lotus Notes, just cited, are also designed for collaborative work.

OR is used for high performance and high availability in large-scale web sites. A recent example is Amazon's "shopping cart," which is designed to be highly available, even if the same user connects to several instances of the Amazon store discussed earlier.

6 Cross references

Consistency Models for Replicated Data. Eventual consistency. Peer-to-peer systems. Traditional Concurrency Control for Replicated Databases. WAN Data Replication.

7 Recommended reading

Optimistic replication, Computing Surveys [8]: a comprehensive survey of OR applications and techniques.

References

[1] Richard A. Golding. *Weak-consistency group communication and membership.* PhD thesis, University of California Santa Cruz, Santa Cruz, CA, USA, December 1992. Tech. Report no. UCSC-CRL-92-52.

[2] Jim Gray, Pat Helland, Patrick O'Neil, and Dennis Shasha. The dangers of replication and a solution. In *Int. Conf. on Management of Data (SIGMOD)*, pages 173–182, Montréal, Canada, June 1996. ACM SIGMOD, ACM Press.

[3] James J. Kistler and M. Satyanarayanan. Disconnected operation in the Coda file system. *ACM Trans. on Comp. Sys. (TOCS)*, 10(5):3–25, February 1992.

[4] Leslie Lamport. Time, clocks, and the ordering of events in a distributed system. *Communications of the ACM*, 21(7):558–565, July 1978.

[5] Gérald Oster, Pascal Urso, Pascal Molli, and Abdessamad Imine. Proving correctness of transformation functions in collaborative editing systems. Research Report RR-5795, LORIA – INRIA Lorraine, December 2005.

[6] K. Petersen, M. J. Spreitzer, D. B. Terry, M. M. Theimer, and A. J. Demers. Flexible update propagation for weakly consistent replication. In *Symp. on Op. Sys. Principles (SOSP)*, pages 288–301, Saint Malo, October 1997. ACM SIGOPS.

[7] Nuno Preguiça, Marc Shapiro, and Caroline Matheson. Semantics-based reconciliation for collaborative and mobile environments. In *Int. Conf. on Coop. Info. Sys. (CoopIS)*, volume 2888 of *Lecture Notes in Comp. Sc.*, pages 38–55, Catania, Sicily, Italy, November 2003. Springer-Verlag.

[8] Yasushi Saito and Marc Shapiro. Optimistic replication. *Computing Surveys*, 37(1):42–81, March 2005.

[9] Chengzheng Sun and Clarence Ellis. Operational transformation in real-time group editors: issues, algorithms, and achievements. In *Int. Conf. on Computer-Supported Cooperative Work (CSCW)*, page 59, Seattle WA, USA, November 1998.

[10] Douglas B. Terry, Alan J. Demers, Karin Petersen, Mike J. Spreitzer, Marvin M. Theimer, and Brent B. Welch. Session guarantees for weakly consistent replicated data. In *Int. Conf. on Para. and Dist. Info. Sys. (PDIS)*, pages 140–149, Austin, Texas, USA, September 1994.

[11] Douglas B. Terry, Marvin M. Theimer, Karin Petersen, Alan J. Demers, Mike J. Spreitzer, and Carl H. Hauser. Managing update conflicts in Bayou, a weakly connected replicated storage system. In *15th Symp. on Op. Sys. Principles (SOSP)*, pages 172–182, Copper Mountain, CO, USA, December 1995. ACM SIGOPS, ACM Press.

[12] Gene T. J. Wuu and Arthur J. Bernstein. Efficient solutions to the replicated log and dictionary problems. In *Symp. on Principles of Dist. Comp. (PODC)*, pages 233–242, Vancouver, BC, Canada, August 1984.

[13] Haifeng Yu and Amin Vahdat. The costs and limits of availability for replicated services. In *18th Symp. on Op. Sys. Principles (SOSP)*, pages 29–42, Lake Louise, AB, Canada, October 2001.